A Hybrid DVS Scheme for Interactive 3D Games

Yan Gu Samarjit Chakraborty
Department of Computer Science
National University of Singapore
E-mail: {guyan, samarjit}@comp.nus.edu.sg

Abstract

Interactive 3D games are now widely available on a variety of mobile devices for which battery-life is a major concern. Many of these devices support voltage/frequency-scalable processors and dynamic voltage scaling (DVS) has emerged as a powerful technique for energy management in such devices. Although DVS algorithms have been very successfully applied to video encoding/decoding applications, their use in interactive computer games has not been sufficiently explored so far. In this paper we propose a novel DVS scheme that is specifically directed towards interactive 3D game applications running on battery-operated portable devices. The key to this DVS scheme lies in an accurate prediction of the rendering workload of a current game scene. We have applied this scheme to First Person Shooter games (e.g. Quake II) and obtained significant power savings while maintaining high frame rates. Based on the observation that there exist two types of workload variations in such games, we compute the voltage/frequency setting for any game scene using a hybrid combination of two different techniques: (i) adjusting the workload prediction using a control-theoretical feedback mechanism, and (ii) analyzing the graphical objects in the current game scene by parsing the corresponding frame. Our scheme is significantly different from those commonly applied to video decoding applications (where only technique (i) is used) and has shown very encouraging results when evaluated with different setups (e.g. laptop running Windows, PDA running Windows Mobile and a configurable simulation platform).

1 Introduction

Over the last few years, graphics-intensive computer games have spilled over from desktop computers to a wide variety of portable devices such as notebooks, PDAs, mobile phones and portable playstations (e.g. see www.doompda.com). Battery-life is one of the major concerns in the design of such devices and has motivated a lot of recent work on dynamic voltage scaling (DVS) [3, 6] and dynamic power management (DPM) [1] schemes. DVS relies on changing the frequency and voltage of the processor at runtime to match the workload demand generated by the application. This has become hugely popular due to the wide availability of dynamic voltage and frequency-scalable processors (e.g. Intel's Pentium Mobile family). On the other hand, DPM-based techniques rely on switching off parts of a device (processor, hard disk, display, etc.) at runtime, based on their usage.

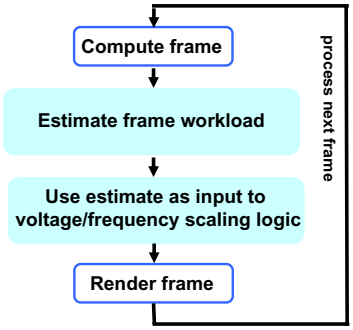


Figure 1. DVS in a game loop.

Although DVS algorithms have been extensively applied to video encoding/decoding applications (which have almost attained the status of the *dining philosophers problem* in this domain) [12, 15, 16], their use in graphics-intensive games has not been sufficiently explored so far. We initiated a study of this problem in [5], where we asked whether game applications are amenable to DVS, and whether known DVS algorithms for video decoding can be applied to games. Our study[1] showed that game applications exhibit sufficient variability in their workload to meaningfully exploit DVS schemes for power savings. Moreover, they offer the possibility of developing DVS algorithms that better exploit the characteristics of game applications (compared to those that have been developed for video decoding). The main differences between game and video decoding applications stem from (i) the *interactive* nature of games, (ii) unlike video frames, game frames cannot be buffered (buffering is exploited in many DVS algorithms, see for example [7]), and (iii) game frames are more "structured" than video frames (which only contain the I, B, or

[1] Based on the publicly availably source code of the Quake II game engine, which also forms the core of many other popular First Person Shooter games like Hexen II.

1080-1812/08 $25.00 © 2008 IEEE
DOI 10.1109/RTAS.2008.33

P frame-type information). More specifically, the workload associated with processing a game frame depends on the contents of the frame, or the constituent *objects*, which can be easily determined by parsing the frame. Similar conclusions were also arrived at in [10, 11] for workload characterization of a 3D graphics processing pipeline.

Contributions of this paper: A game application, similar to a video decoder, runs in an infinite loop and processes a sequence of game frames that have to be rendered and displayed on the screen. Our proposed DVS algorithm for such applications is integrated into this game loop. The critical step is to accurately predict the workload of a frame, which is followed by a "voltage/frequency scaling logic" (that we describe in more detail later in this paper). An overview of this DVS algorithm is shown in Figure 1.

As an initial attempt, we developed a DVS algorithm in which each game frame is parsed to identify its constituent objects, based on which the workload associated with rendering the frame is estimated. This estimate, along with the target frame rate is used to determine the frequency (and voltage) of the processor running the game application.

Note that this scheme (which we henceforth refer to as *frame structure-based workload prediction*) is fundamentally different from the control-theoretic feedback schemes commonly used in DVS algorithms for video decoding [8, 9, 14], where previous prediction errors are taken into account while estimating the workload of a current frame. Although our frame structure-based prediction works well (and outperforms control-theoretic prediction schemes) for game plays where the frame workload exhibits sufficient variability, often there are sequences of frames with relatively constant rendering workload. For such frames, control-theoretic prediction schemes happen to perform better. To take advantage of both these schemes, in this paper we propose a *hybrid* workload prediction scheme, where we keep on switching between the two schemes based on their relative performance.

In the hybrid workload prediction scheme, the workload associated with rendering a game frame is roughly the sum of the workloads generated by processing the different objects (e.g. *brush models*, *alias models*, etc.) constituting the frame. Each of these workload components is estimated separately and then summed up to compute the processing workload of a frame. Whenever the estimation error for any object type is beyond a certain level, the scheme switches to a different prediction mode for that object type. In other words, our hybrid scheme is applied at an *object* level rather than at the frame level. Naturally, this leads to a more accurate workload prediction (albeit at the cost of slightly higher computational overhead) compared to applying this scheme at the frame level.

An illustrative example: Figure 2 shows the workload variation for the *particles* object type in an excerpt from a

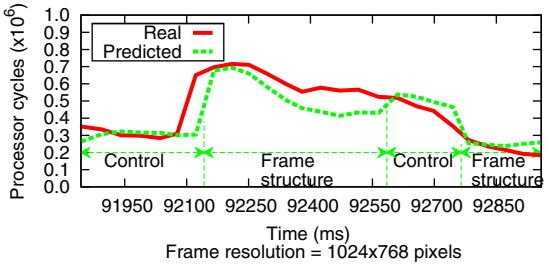


Figure 2. Sample run of the hybrid scheme.

game play, and the corresponding run of our workload prediction scheme. Note that the first switch from the control-theoretic scheme to the frame structure-based prediction scheme occurs when the workload (measured in terms of number of processor cycles) exhibits a significant increase from a flat profile. Once the profile again becomes relatively flat, the relative performance of the frame structure-based scheme degrades and the scheme switches back to the control-theoretic scheme. The final change from the control-theoretic to the frame structure-based scheme again occurs when the workload profile shows a relatively large dip. In summary, this scheme keeps on switching between the two workload prediction schemes based on their relative prediction errors. For a relatively flat workload profile, as seen in Figure 2, the prediction accuracy of the control-theoretic scheme dominates. Whenever the workload profile exhibits a significant variation, the frame structure-based prediction scheme takes over.

Organization of the paper: The rest of this paper is organized as follows. The next section gives an overview of the architecture of a game engine. Section 3 outlines the workload prediction scheme using a PID controller. In Section 4 we describe the frame structure-based workload prediction technique, followed by our hybrid workload prediction scheme in Section 5. The details of using this scheme for dynamic voltage and frequency scaling are discussed in Section 6. We evaluate the performance of our hybrid DVS scheme in Section 7 and finally we conclude in Section 8 by outlining some directions for future work.

2 The Game Engine

As shown in Figure 1, a game engine runs in an infinite loop, where the loop body consists of tasks responsible for processing a single game frame. These tasks can be categorized as *computing* and *rendering*. Examples of computing tasks are collision detection, AI, simulation of game physics and particle systems. Rendering tasks implement algorithms to generate an image or a frame from a model, which is then displayed on the screen (see [2, 13] for more details).

Rendering algorithms typically transform vertices of 3D/solid objects onto the 2D screen space, delete invisible pixels by clipping, perform rasterization, delete occluded pixels and interpolate various parameters. These algorithms

are computationally expensive and are typically mapped onto a graphics processing unit (GPU) in desktop computers or high-end laptops. However, most battery-powered mobile devices (e.g. PDAs, cell phones and portable gaming devices) currently do not have GPUs and instead implement the rendering algorithms in software. We believe that this trend will continue at least for the next few years. Hence, for this work we have assumed all rendering tasks to be implemented in software running on the common voltage/frequency-scalable processor along with the computing tasks of the game engine.

3 Control Theoretic Workload Prediction

As we mentioned above, although control-theoretic feedback mechanisms have been widely used for DVS in video decoding applications, they have not been explored in the context of games. More importantly, the buffer-centric approaches for workload prediction in video decoding applications cannot be applied in this context since game frames are not buffered (because of the interactive nature of game applications). In this section, we propose a control-theoretic workload prediction scheme for interactive game applications, which constitutes one of the two components of our hybrid scheme.

3.1 PID Controller Basics

A PID controller is a generic control loop feedback mechanism that is used to adjust system parameters based on the feedback from the recent error between a *measured process variable* and a desired *set point*. It involves three separate components – *Proportional Control*, *Integral Control* and *Derivative Control*. The proportional control determines the speed of the system in reacting to errors. The integral control is used to determine the accuracy of the system based on recent errors. Finally, the derivative control determines the system reaction based on the rate at which the error changes. Such a controller can be implemented in several ways, the easiest among which is the parallel form where each control element is given the same error input in parallel. The output of the controller is given by $Output(t) = P_{contrib} + I_{contrib} + D_{contrib}$, where $P_{contrib}$, $I_{contrib}$ and $D_{contrib}$ are the feedback contributors of the PID controller. $P_{contrib} = K_p \cdot \epsilon(t)$, $I_{contrib} = \frac{1}{I} \cdot \int \epsilon(t)dt$ and $D_{contrib} = D \cdot \frac{d\epsilon(t)}{dt}$, where $\epsilon(t)$ is defined as the difference between a measured variable and a desired set point. K_p, I and D are the proportional, integral and derivative constants. By tuning the values of these constants the PID controller can provide individualized control, specific to process requirements involving error responsiveness, overshoot of the set point and system oscillation.

3.2 A PID-based Workload Predictor

Here, the goal is to *predict* the workload of a game frame before the frame is processed/rendered. This is estimated to be the sum of the predicted workload of the previous frame and the output of the PID controller, which takes as input the prediction errors of a certain number of previously processed frames. Towards this we select the predicted workload $\bar{\omega}_i$ as the *measured variable* and the actual workload ω_i as the *set point*. The resulting error is periodically measured by the PID controller and is given by $\epsilon(t) = \omega_i - \bar{\omega}_i$, where t is the time stamp of the i-th frame. The following discrete PID controller formulation is used in our DVS scheme:

$$\Delta\bar{\omega}_i = K_p \cdot \epsilon(t) + \frac{1}{I} \cdot \sum_{T_I} \epsilon(t) + D \cdot \frac{\epsilon(t) - \epsilon(t - T_D)}{T_D} \quad (1)$$

$$\bar{\omega}_{i+1} = \bar{\omega}_i + \Delta\bar{\omega}_i \quad (2)$$

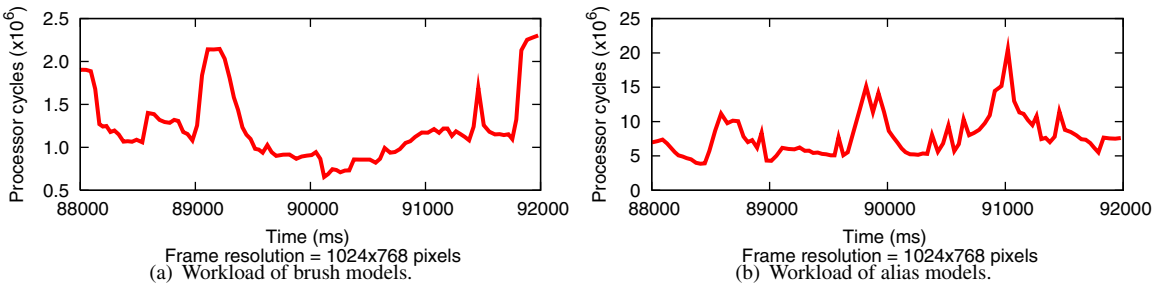
As we explained in Section 3.1, here K_p, I and D are the proportional, integral and derivative coefficients respectively. T_I and T_D are the tunable parameters of the controller. In our scheme, T_I is set to the frame interval, and hence, $\sum_{T_I} \epsilon(t)$ is the sum of the prediction errors from frame $(i - T_I)$ to frame i. T_I is set to be 5 frames; hence, only the errors from previous 5 frames contribute to the integral element. T_D is set to be equal to the frame execution time and hence $\epsilon(t - T_D)$ is the prediction error at time $t - T_D$. We restrict T_D to be the execution time of the last one frame to ensure that multiple feedback corrections do not affect one another. The output $\Delta\bar{\omega}_i$ is fed back to the controller and regulates the next estimated frame workload $\bar{\omega}_{i+1}$.

In our experiments, we observed that the values of the PID controller parameters have a significant impact on the prediction results. By manually tuning these parameter values, we obtained the best results when $K_p = 0.5$, $I = 28$ and $D = 0.00001$ for the entire game play. However, the optimal choice of these values might vary from one game engine to the next (depending on the variability in the resulting workload).

In our PID controller-based DVS scheme, the value of the error $\epsilon(t)$ (obtained from previous frames) is used to compute the proportional, integral and derivative elements. Finally, the sum of these elements is used to estimate the workload of the current frame. This is followed by computing the voltage/frequency of the processor based on the predicted frame workload and the target frame rate. The processor's voltage/frequency is then scaled based on a *scaling logic* followed by the rendering steps.

4 Workload Prediction via Frame Structures

In this section we describe the frame structure-based workload prediction scheme that forms the second of the two components of our hybrid DVS algorithm. It may be noted that a significant component of the rendering task involves *rasterizing* objects on the screen. Our experimental results suggest that the total workload generated from processing a frame is almost linearly correlated with its rasterization workload. Hence, we predict the total workload by estimating the rasterization workload of a frame.



(a) Workload of brush models. (b) Workload of alias models.

Figure 3. Rasterization workload variations for brush models and alias models.

4.1 Exploiting Frame Structures

The proposed frame structure-based workload prediction scheme primarily consists of estimating the rasterization workload for each frame. Towards this, we compute the number of occurrences of the different primitives in a frame (e.g. brush models, alias models and particles) and multiply these with the workload involved in processing each of these primitives. This is possible because, once again, the workload involved in processing all primitives of the same type almost linearly scales with the number of primitives occurring in the frame. The workload corresponding to a single primitive of any given type is computed in an offline fashion. We have experimentally verified this (nearly) linear correlation for most of the primitives such as brush models, alias models, textures and particles (see [5]). It may also be noted that the workload corresponding to each of these primitives exhibits sufficient short-term variability as shown in Figure 3 (because of the varying number of occurrences of these primitives in a sequence of consecutive frames). Hence, in the presence of high workload variations, this scheme performs significantly better than control-theoretic predictors when applied individually to the different primitive-types. In what follows, we describe our rasterization workload estimation in further details.

For each frame, once the current *view frustum* is computed based on the user input, the number of occurrences of the different primitives in the frame is estimated (e.g. the number of brush models, alias models, etc.). Further, for each of these primitives, its detailed constitution is also computed. For each brush model, this amounts to computing its number of constituent polygons. For each alias model it amounts to computing its number of pixels, for each texture model its number of surfaces, and for each particle its number of pixels. Based on offline simulation, the workload associated with each of the different primitives parameterized by their constitution is stored in a table. For example, this table contains the workload associated with processing a brush model with n polygons for different values of n. Let us assume that $c(n)$ is the number of processor cycles required to rasterize any *one* brush model with n polygons. Then $c(n)$ is stored in the above-mentioned table. To compute the rasterization workload for *all* brush models in a frame, let us assume that $B(n)$ is the number of brush models in this frame with n polygons. Then the total rasterization workload for all the brush models in the frame is equal to $\sum_{n=1,\ldots,\infty} c(n) \times B(n)$ processor cycles (where ∞ is the maximum possible number of polygons in any brush model). This procedure is followed for all the different primitives, with the exception of texture models (which is explained below).

The abovementioned estimation process clearly has a computational overhead, which turns out to be prohibitive in the case of textures. Textures are drawn on brush models and hence their number of constituent surfaces cannot be determined unless the associated brush models are rasterized. However, rasterizing brush models solely for workload estimation purposes is prohibitively expensive. Hence, the workload associated with rasterizing texture models in a frame is estimated using a control-theoretic prediction scheme. Although this is not as accurate as frame structure-based predictions, this loss of accuracy is unavoidable. However, using frame structure-based estimation, at least for the other primitives like brush models, alias models and particles, reduces the overall error compared to using control-theoretic predictors for all the primitives. Further, this mix of two different estimation schemes results in a good tradeoff between prediction accuracy and computational overhead.

5 A Hybrid Workload Prediction Scheme

In this section, we present our hybrid frame workload prediction which combines the two techniques described above, i.e. (i) adjusting the workload prediction using a control-theoretic feedback mechanism (viz. PID controller), and (ii) analyzing the graphical objects in the current game scene by parsing the corresponding frame (viz. the frame structure scheme).

5.1 Workload Variation

As we already discussed, game workload exhibits a large degree of variability. We have experimented with various game plays and workload prediction schemes. In particular, we have investigated three different prediction schemes: (i) history-based predictors, where the workload of a game frame is estimated to be the average of the workloads of

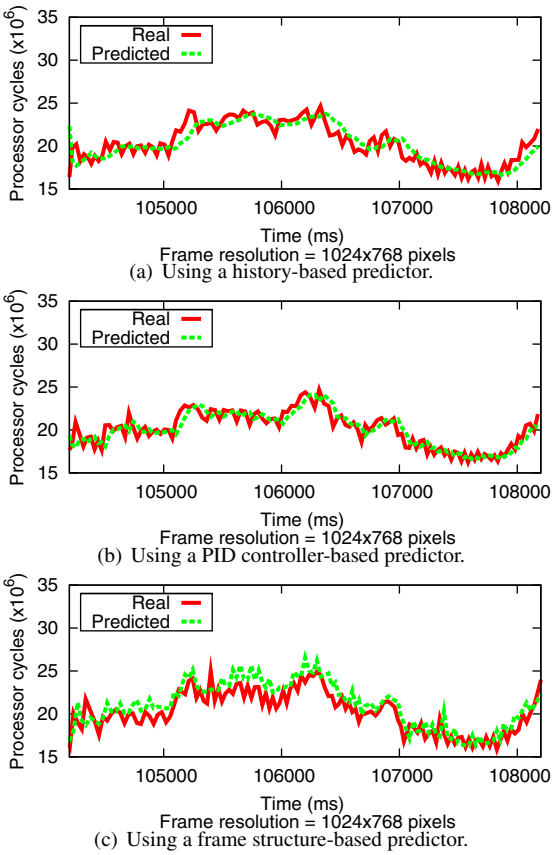
a certain number of previous frames, (ii) PID controller-based predictors (as described in Section 3), and (iii) frame structure-based predictors (as described in Section 4). Our results show that for frame sequences which exhibit relatively low workload variation, the first two schemes outperform (iii). However, there also exists frame sequences with high workload variability, and for such sequences a frame structure-based predictor is certainly better. In fact, this observation is the main motivation behind devising a hybrid workload prediction scheme.

Figure 4 shows a comparison of the three abovementioned schemes for a sequence of game frames with relatively low variability. Clearly, the PID controller-based prediction scheme outperforms the other two, with the frame structure-based predictor giving the worst result. Figure 5 shows the same comparison for a sequence of frames exhibiting a high variability in their workload. Here, Figures 5(a), (b) and (c) show the performance of the history, PID controller, and frame structure-based predictors. It is easy to see that for this sequence of frames, the frame structure-based predictor outperforms the first two. Figures 5(d) and (e) show the performance of two hybrid predictors, which perform even better. We describe the details of these two predictors in the following sections.

5.2 Prediction Mode Switching

As mentioned before, the total workload involved in processing a frame is made up of the sum of the rasterization workloads of the different primitives that constitute the frame and the workload associated with tasks such as collision detection, AI, simulation of game physics and particle systems. Our prediction scheme estimates the rasterization workload and scales it appropriately to obtain the total frame workload (which is almost linearly proportional to the rasterization workload; see Section 4.1). Notice that the same workload estimation scheme is executed for each type of primitives (i.e. brush models, alias models and particles), with the exception of textures (for reasons which were explained in Section 4.1).

In what follows, we describe how to switch between the two prediction schemes (viz. the PID controller-based scheme and the frame structure-based scheme). First, it should be noted that when the frame structure-based scheme is in use, the PID controller-based scheme is also kept active, but its estimation result is not used for voltage/frequency scaling. This incurs a certain (negligible) computational overhead. However, when the PID controller-based scheme is in use, the frame structure-based predictor is switched off because it is computationally expensive. If the workload of the last frame is estimated using the frame structure-based predictor, then our proposed algorithm computes two *prediction errors*; one incurred by the frame structure-based predictor and the other incurred by the PID controller. If the former error is larger than the latter



(a) Using a history-based predictor.

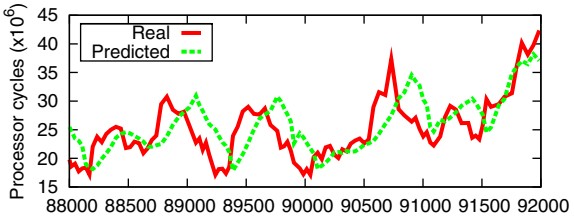
(b) Using a PID controller-based predictor.

(c) Using a frame structure-based predictor.

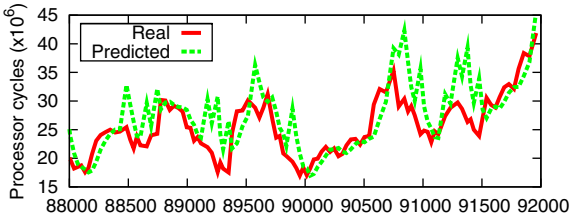
Figure 4. Workload prediction using three different schemes for a frame sequence with relatively low workload variability.

then a prediction mode switch is enabled (i.e. the scheme switches to the PID controller-based predictor), otherwise the same predictor is retained.

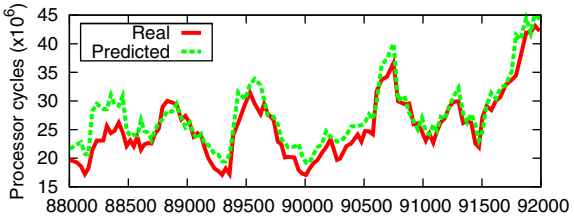
On the other hand, if the PID controller is currently in use, the mode switching decision is based on whether the prediction error for the last frame is greater than a certain *threshold error*. However, this threshold error is not statically predefined, but is constantly computed (or updated) at runtime. This threshold error depends on the frames whose workload was last predicted using the frame structure-based predictor. Let these be the α-th to the β-th frames. In other words, α and β are such that the workload of the $(\alpha-1)$-th and $(\beta+1)$-th frames were predicted using the PID controller-based predictor, and the workload of the α-th to the β-th frames were predicted using the frame structure-based predictor. The *threshold error* is equal to $\min(\frac{\sum_{i=\alpha}^{\beta}\epsilon_{i,o}}{\beta-\alpha+1}, \frac{\sum_{i=\alpha}^{\beta}\epsilon_{i,c}}{\beta-\alpha+1}) + \tau|\frac{\sum_{i=\alpha}^{\beta}(\epsilon_{i,o}-\epsilon_{i,c})}{\beta-\alpha+1}|$, where $\epsilon_{i,o}$ and $\epsilon_{i,c}$ denote the prediction errors of the i-th frame as incurred by the frame structure-based and the PID controller-based schemes respectively. The value of $0 < \tau \leq 1$ has to be appropriately chosen. With the PID controller currently in use, if the prediction error of a frame exceeds the *threshold error*, then a mode switch is enabled.



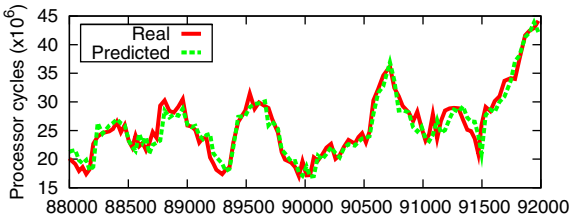
(a) Using a history-based predictor.



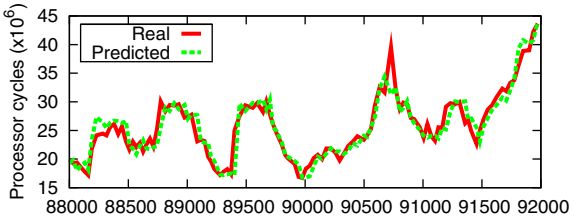
(b) Using a PID controller-based predictor.



(c) Using a frame structure-based predictor.



(d) Using a history+frame structure-based hybrid predictor.



(e) Using a PID controller+frame structure-based bybrid predictor.

Figure 5. Workload prediction using five different schemes for a frame sequence exhibiting high workload variability.

Once again, note that this scheme is individually applied to all the different primitives constituting a frame, except for textures, whose workload is always estimated using a PID controller-based scheme. The total estimated workload of a frame, along with the desired frame rate is used to compute the target processor frequency, which is then mapped onto a discrete voltage/frequency level that is supported by the processor. In summary, our hybrid predictor switches between two prediction schemes based on their relative performance. Finally, it may be noted that the PID controller-based predictor may be replaced by a simple history-based predictor (that was described in Section 5.1) with everything else remaining the same. This would result in a history + frame structure-based hybrid predictor.

5.3 Prediction Accuracy and Overheads

In this section we compare the performance of the following predictors in terms of the overhead incurred and the prediction accuracy.

- `History`: History-based predictor that estimates the workload of a frame by averaging the actual workload of a certain number of previously processed frames.
- `PID Controller`: The PID controller-based predictor described in Section 3.
- `Frame structure`: The frame structure-based predictor described in Section 4.
- `Hybrid(history)`: The hybrid predictor that switches between `History` and `Frame structure` predictors.
- `Hybrid(control)`: The hybrid predictor that switches between `PID Controller` and `Frame structure` predictors.

5.3.1 Prediction Overhead

As one would expect, both `History` and `PID Controller` incur negligible computational overheads. The `Frame structure` workload predictor is computationally more expensive and incurs, on an average, 1.7 million processor cycles per frame on a laptop with an Intel Pentium Mobile processor running Windows XP. Note from Figure 5 that the workload generated by processing a frame varies between 15 to 45 million cycles. Hence, the computational overhead incurred by the frame structure-based predictor and its hybrid combinations is certainly within the feasible region.

5.3.2 Prediction Accuracy

Figure 5 compares the different workload predictors against the actual workload of a sequence of game frames. The excerpt shown in this figure was generated from a 4 sec demo file of Quake II (massive1.dm2[2]) running on a laptop with Windows XP. Each point in Figure 5 corresponds to a frame and the horizontal axis refers to the time stamp (in milliseconds) associated with each frame. The vertical axis refers to the total processing workload of a frame in terms of the number of processor cycles. It may be noted that the workload varies between 15 million cycles to 45 million cycles per frame (and therefore offers the possibility of dynamic voltage/frequency scaling).

[2]http://cure.gamepoint.net/files/massive1.zip

It it clear from this figure that our proposed hybrid schemes match the profile of the actual frame workload more closely than the history, PID controller, or frame structure-based predictors. To measure the incurred prediction error, we used two metrics: (i) the *absolute prediction error* which is defined as the absolute difference in processor cycles between the actual and predicted workloads, and (ii) the *relative prediction error* which is defined as the ratio between the absolute prediction error and the actual workload. Note that the errors in Figure 5(a) (i.e. using the `History` predictor) turn out to be 3.6 million cycles and 0.15 respectively. These errors drop to 1.2 million cycles and 0.05 respectively using our proposed `Hybrid(control)` scheme (see Figure 5(e)).

We observed the similar results obtained by running a 10 sec demo file of Quake on a PDA with Windows Mobile. The reason behind choosing Quake (instead of Quake II) is that the high computational workload generated by Quake II results in unacceptably low frame rates on a PDA, thereby deteriorating the game quality. The workload on the PDA varies between 20 million processor cycles and 90 million cycles per frame. The absolute and relative errors by using the `History` predictor are 6.2 million cycles and 0.15 respectively. These errors drop to 2.5 million cycles and 0.06 respectively for the `Hybrid(control)` predictor. Notice that on both the platforms, our proposed `Hybrid(control)` predictor results in more than 60% improvement in prediction accuracy over a simple history-based workload predictor.

6 Dynamic Voltage/Frequency Scaling

In this section we discuss how the above workload prediction scheme is integrated with a DVS algorithm. As shown in Figure 1, the predicted workload for each frame is fed into a voltage/frequency scaling logic, which takes into account several hardware and systems related issues to decide whether the voltage/frequency should be switched from its current level.

Frequency transitions on any processor always incur an overhead which depends on the processor's microarchitecture and the operating system running on top of it. For example, the average frequency transition overhead for Windows XP running on an Intel Pentium Mobile processor is 20 million cycles, which is equivalent to 14 milliseconds with the operating frequency set to 1400 MHz.

In order to avoid excessive frequency transitions, which might defeat the purpose of using a DVS algorithm, we have used a *lazy* transition mechanism. Instead of triggering a frequency switch with every possible frame-workload change, our lazy mechanism defers the frequency switch by one frame. Let f_i be the processor frequency with which $frame_i$ is processed. Let f_{i+1} be the frequency computed by any of our workload predictors for $frame_{i+1}$, where $f_i \neq f_{i+1}$. Instead of switching the processor frequency to f_{i+1}, our frequency-scaling logic defers the switch to $frame_{i+2}$. If the computed frequency f_{i+2} is equal to f_{i+1}, then the processor's frequency is switched to f_{i+2}, else the same procedure is followed. Such a lazy frequency transition mechanism cuts down excessive frequency transitions, thereby reducing the transition overhead to a large extent. Note that although we defer the transition decision by only one frame, depending on the transition overhead on a different platform, this decision can also be deferred by multiple frames. Finally, note that we switch the operating frequency of the processor with the assumption that the voltage is automatically scaled accordingly (i.e. we do not explicitly control the operating voltage of the processor).

7 Experimental Evaluation

We have evaluated our proposed DVS scheme by integrating it with the Quake/Quake II game engines running on a number of different platform settings: (i) on a laptop with an Intel Pentium Mobile processor running Windows XP, (ii) on a PDA with an Intel XScale processor running Window Mobile 5, (iii) using a discrete event simulator where the processor has the same power consumption characteristics as in the laptop/PDA, but its frequency transition overhead is assumed to be zero, and (iv) same as (iii) with the additional assumption that the processor's frequency is continuously scalable. Settings (iii) and (iv) are referred to as *simu-disc* (i.e. simulation with discrete frequency levels) and *simu-cont* (simulation with continuous frequency levels) respectively. These two settings represent ideal cases and the results obtained using them give an upper bound on the power savings that can be obtained using our scheme.

Our motivation behind using the Quake-serial game engines primarily stems from the fact that it is a popular game that can be played on a variety of mobile devices such as PDAs, mobile phones and laptops without additional graphics hardware. Further, this game engine forms the core of a number of other First Person Shooter games (e.g. Hexen II) and its software architecture is representative of those in many other commercially-available games. Finally, the source codes of Quake and Quake II are freely available, which allows for experimentation and appropriate modification.

To ensure reproducibility, we have used pre-recorded demo files. Since these demo files keep pre-recorded states and therefore they are not computed during playback, there is some difference in workload when compared to a real-time game play. However, we have verified that these differences are negligible and do not affect the conclusions derived from this study. Finally, the game resolution on the laptop was set to 1024×768 pixels, running in full-screen mode. Again, the conclusions derived from this setting also hold for other resolutions, as verified by our experiments

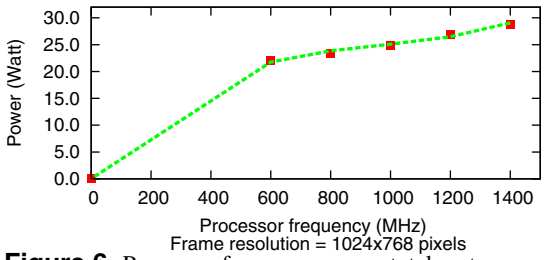


Figure 6. Processor frequency versus total system power consumption on a laptop.

with running Quake at 240×320 pixels on the PDA. To ensure that the game process is not preempted by other processes, it was set to the highest priority.

Laptop Settings: The laptop used for our experiments (with the 1400 MHz Intel Pentium Mobile processor) was equipped with Speedstep technology and had an ATI Radeon Mobility Video card. The processor supported five different operating points with clock frequencies of 1400, 1200, 1000, 800 and 600 MHz. For Pentium processors, the *RDTSC*[3] (read time-stamp counter) instruction is an excellent high-resolution, low-overhead mechanism to collect execution requirements of tasks in terms of processor cycles. All processor cycle counts on the laptop were measured using the *RDTSC* instruction that was inserted into the Quake II source code.

All the power measurements were conducted by connecting this laptop to a National Instruments PXI-4071 $7\frac{1}{2}$-digit Digital Multimeter, using which the instantaneous voltage and current drawn by the laptop were recorded. Our estimated power consumptions therefore refer to the full system power and not that of the processor alone. Figure 6 shows the total system power consumption for the five different processor frequency levels. It may be noted that this varies between 28.8 Watts and 22.1 Watts, which correspond to the processor frequencies of 1400 MHz and 600 MHz respectively. Hence, the maximum possible reduction in power consumption is upper bounded by 23%.

PDA Settings: The PDA used for our experiments was a Dell Axim X51 with a 520 MHz Intel XScale PXA270 processor and 64MB SDRAM. The processor supports six different operating frequency points: 520, 416, 312, 208, 156 and 104 MHz. Unfortunately, the *RDTSC* instruction is not supported by XScale processors and processor cycle counts cannot be read by application programs. Hence, we used high-resolution and low-overhead Windows APIs such as the *QueryPerformanceCounter* to retrieve the processor time associated with different tasks. However, the results returned by these APIs become unreliable especially when the operating frequency of the processor is changed at runtime. To avoid these problems we, in addition, conducted experiments using a discrete event simulator with the power characteristics measured from the PDA.

To estimate the power characteristics of the PDA, we measured the power consumption of its CPU-core by connecting an iWave prototype PDA board[4] to the National Instruments PXI-4071 $7\frac{1}{2}$-digit Digital Multimeter (as we did with the laptop). The iWave prototype board has the same processor as many regular PDAs (i.e. Intel XScale PXA270). Furthermore, each component on the board (e.g. CPU-core, LCD, wireless interface, etc.) can be hooked up to measuring instruments. Hence, the measured CPU-core power consumption of the iWave board was used to estimate the power characteristics of a regular PDA. We observed that the power consumption of the CPU-core varies between 0.4 to 0.13 Watt, corresponding to the frequency range 520 - 104 MHz. Therefore, the maximum possible reduction in power consumption is upper bounded by 68%. This clearly shows that DVS can achieve much better power savings when the CPU-core of the PDA is considered in isolation, compared to the maximum system-wide power savings that can be achieved for the laptop we experimented with (which, as mentioned above, is 23%).

7.1 Results

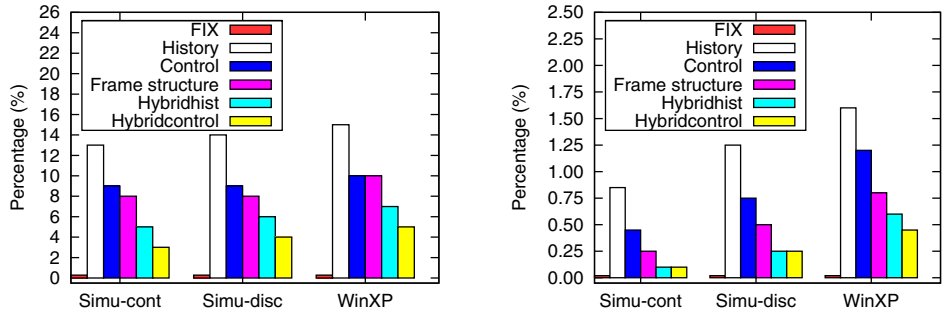
We have defined two quality metrics that have been motivated by a study in [4]. This study concluded that while frame rates higher than a pre-defined constant target frame rate do not improve the overall gaming experience, lower than target frame rates severely degrade the game quality. Our first metric only measures the percentage of frames that miss their deadlines. The second metric also takes into account the magnitude of the missed deadlines (or the *tardiness*).

We have compared the performance of different DVS schemes: `FIX` (where the processor is run at a constant frequency of 1400 MHz, i.e. no frequency scaling), `History` (DVS with a history-based predictor), `PID Controller` (DVS with the PID controller-based predictor), `Frame structure` (DVS using the frame structure-based workload prediction scheme), `Hybrid(history)` (DVS with a hybrid combination of history-based and the frame structure-based predictors), and `Hybrid(control)` (DVS with our proposed hybrid combination of a PID controller-based predictor and a frame structure-based predictor). For all our experiments on the laptop, we have set the target frame rate to 20 frames/second. Hence, each frame has to be processed within $1/20$th of a second, which is set as the *frame deadline*. We have manually tuned the PID controller parameters in `PID Controller` and `Hybrid(control)` and obtained the best results with $K_p = 0.5$, $I = 28$, and $D = 0.00001$ on the laptop.

Clearly, the energy consumption during a game play and

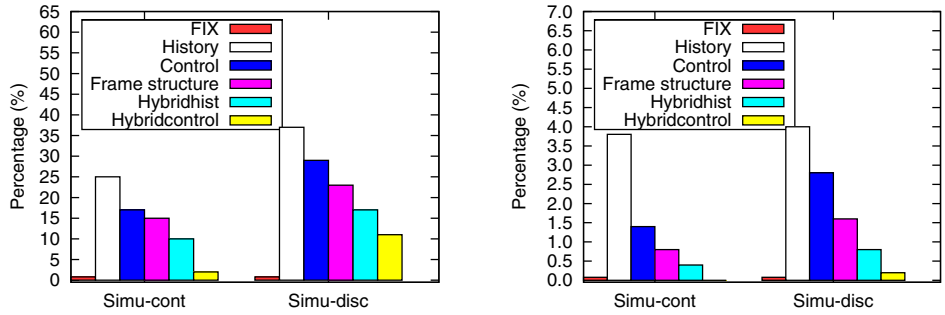
[3] Intel 64 and IA-32 Software Developer's Manual Vol 2B

[4] http://www.iwavesystems.com/



(a) Percentage of frames with missed deadlines. (b) Average tardiness of frames.

Figure 7. Comparison of game quality using different prediction schemes on a laptop running WinXP (with the target frame deadline set to $1/20$th of a second).



(a) Percentage of frames which missed their deadlines. (b) Average tardiness of frames.

Figure 8. Comparison of game quality using different prediction schemes on a PDA (with the target frame deadline set to $1/5$th of a second)

the quality of the game are mutually dependent on each other. Hence, to compare the different DVS schemes (i) we fix the energy consumption and then measure the output quality resulting from the different schemes, and (ii) we fix the output quality (e.g. *all* frames have to be processed within their prespecified deadline, which is equal to $1/20$th of a second when the target frame rate is 20 frames/sec) and measure the energy consumption resulting from the different schemes.

Figure 7 shows the game quality for the different DVS schemes under the two metrics outlined above. From this figure, it may be noted that under the average tardiness metric, our proposed `Hybrid(control)` scheme results in more than 72% improvement over `History` on a laptop running Windows XP, for the same amount of energy consumption. The results under the simulation setting (with the PDA power characteristics) are even more attractive. In terms of power savings, compared to the `FIX` scheme, our proposed `Hybrid(control)` scheme achieves up to 22% power savings, where the upper bound on the savings, as mentioned before, is 23% on the laptop. Note that to match the target frame deadline, most of the frequencies computed for the estimated frame workload approach the lowest possible frequency (i.e. 600 MHz) on the laptop.

When the target frame deadline is reduced to $1/30$th of a second (i.e. 30 frames/sec), more processor cycles are required to speed up the game play. Therefore, the power savings from `Hybrid(control)` drops to 13% comparing with `FIX`, while the game quality obtained using the `Hybrid(control)` is consistently better than that obtained using `History`.

We also evaluated the power consumptions resulting from the different DVS algorithms on the laptop, when *all* the frames are required to meet their deadlines. We observed that the power savings of `Hybrid(control)` over `History` on the laptop are not as obvious as the improvements in quality as discussed above. This is primarily because we considered the power consumption of the entire laptop and not the CPU alone. As shown in Figure 6, only around 6% more power is consumed, even when the frequency is scaled to one higher level on the laptop.

We also conducted similar experiments with a longer demo file (160 sec) and observed that under the average tardiness metric, our proposed `Hybrid(control)` scheme results in more than 29% improvement over `History` for the same amount of energy consumption. In terms of power saving, compared to the `FIX` scheme, our proposed `Hybrid(control)` scheme achieves around 21% power savings.

7.2 Results on the PDA

On the PDA, the best results were obtained with $K_p = 0.7$, $I = 50$, and $D = 0.00001$ for the target frame rate set to 5 frames/second. For the `FIX` scheme, the XScale processor is run at a constant frequency of 520 MHz. Figure 8 shows the game quality under the two metrics for the different DVS schemes on the PDA. Note that under the average tardiness metric, our proposed `Hybrid(control)` scheme leads to more than 100% quality improvement over `History` with the *simu-cont* setting for the same power consumption. With the *simu-disc* setting, this drops to 95% improvement. Finally, compared to `FIX`, our scheme yields more than 35% and 25% improvements in power savings with the *simu-cont* and *simu-disc* settings, where the upper bound on the savings, as mentioned before, is 68%.

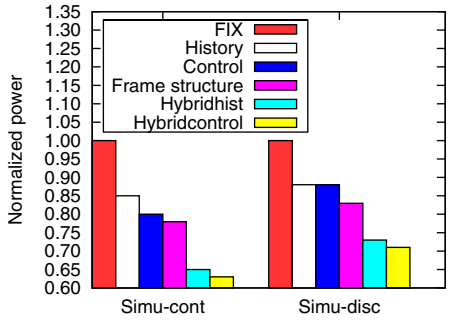


Figure 9. Normalized power consumption using the different prediction schemes on the PDA.

As explained before, the CPU-core power consumption exhibits a relatively large change as the processor frequency is scaled. Therefore, the power savings of `Hybrid(control)` over `History` on the PDA are more significant than those on the laptop, when no frames are allowed to miss their predefined deadlines. Figure 9 shows the CPU-core power consumptions resulting from the different DVS algorithms using the simulation platform with PDA power characteristics. Here, `Hybrid(control)` achieves 26% and 19% more CPU-core power savings than `History` under the *simu-cont* and *simu-disc* settings respectively. *simu-cont* has more savings than *simu-disc* since we assume continuous frequency scaling and no frequency transition overhead (i.e. it represents an ideal case, which results in an upper bound on the energy savings).

8 Concluding Remarks

We have proposed a hybrid DVS scheme targeted towards graphics-intensive game applications. Our experimental results indicate significant power savings and improved output quality compared to known DVS algorithms on different evaluation platforms (e.g. laptop, PDA, and simulation based). Currently we are in the process of investigating game application-specific power management techniques for other components on a portable device (e.g. wireless interface, LCD, etc.), especially in the context of multiplayer games.

References

[1] A. Acquaviva, L. Benini, and B. Riccó. An adaptive algorithm for low-power streaming multimedia processing. In *Design, Automation and Test in Europe (DATE)*, 2001.

[2] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz. Designing a PC game engine. *IEEE Computer Graphics and Applications*, 18(1), 1998.

[3] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *International Conference on Computer-Aided Design (ICCAD)*, 2002.

[4] M. Claypool, K. Claypool, and F. Damaa. The effects of frame rate and resolution on users playing First Person Shooter games. In *Multimedia Computing and Networking Conference (MMCN)*, 2006.

[5] Y. Gu, S. Chakraborty, and W. T. Ooi. Games are up for DVFS. In *Design Automation Conference (DAC)*, 2006.

[6] C. Huges, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *ACM/IEEE International Symposium on Microarchitecture*, 2001.

[7] C. Im, S. Ha, and H. Kim. Dynamic voltage scheduling with buffers in low-power multimedia applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):686–705, 2004.

[8] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2002.

[9] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Reducing multimedia decode power using feedback control. In *International Conference on Computer Design (ICCD)*, 2003.

[10] B. Mochocki, K. Lahiri, and S. Cadambi. Power analysis of mobile 3D graphics. In *Design, Automation, and Test in Europe (DATE)*, 2006.

[11] B. Mochocki, K. Lahiri, S. Cadambi, and X. S. Hu. Signature-based workload estimation for mobile 3D graphics. In *Design Automation Conference (DAC)*, 2006.

[12] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *ACM Multimedia (MM)*, 2003.

[13] A. Watt and F. Policarpo. *3D Games: Real-time Rendering and Software Technology, Volume 1*. Addison-Wesley, 2001.

[14] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004.

[15] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[16] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. In *ACM Multimedia (MM)*, 2004.